Attorney Docket No.: 7451.0003-02
InterTrust Ref. No.: IT-9.2 (US)

UNITED STATES PATENT APPLICATION

FOR

SYSTEMS AND METHODS USING CRYPTOGRAPHY TO PROTECT SECURE COMPUTING ENVIRONMENTS

BY

VICTOR H. SHEAR

W. OLIN SIBERT

and

DAVID M. VAN WIE

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT
& DUNNER, L.L.P.
STANFORD RESEARCH PARK
700 HANSEN WAY
PALO ALTO, CALIF. 94304
650-849-6600

SYSTEMS AND METHODS USING CRYPTOGRAPHY TO PROTECT SECURE COMPUTING ENVIRONMENTS

CROSS REFERENCE TO RELATED APPLICATION

This application is related to commonly assigned copending application Serial Number 08/388,107 of Ginter et al., filed 13 February 1995, entitled "SYSTEMS AND METHODS FOR SECURE TRANSACTION MANAGEMENT AND ELECTRONIC RIGHTS PROTECTION" (attorney reference number 895-13). We incorporate by reference, into this application, the entire disclosure of this prior-filed Ginter et al. patent application just as if its entire written specification and drawings were expressly set forth in this application.

FIELD OF THE INVENTION(S)

This invention relates to computer security, and more particularly to secure and/or protected computer execution environments. Still more specifically, the present invention relates to computer security techniques based at least in part on cryptography,

that protect a computer processing environment against potentially harmful computer executables, programs and/or data; and to techniques for certifying load modules such as executable computer programs or fragments thereof as being authorized for use by a protected or secure processing environment.

BACKGROUND AND SUMMARY OF THE INVENTION(S)

Computers have become increasingly central to business, finance and other important aspects of our lives. It is now more important than ever to protect computers from "bad" or harmful computer programs. Unfortunately, since many of our most critical business, financial and governmental tasks now rely heavily on computers, dishonest people have a great incentive to use increasingly sophisticated and ingenious computer attacks.

Imagine, for example, if a dishonest customer of a major bank could reprogram the bank's computer so it adds to instead of subtracts from the customer's account — or diverts a penny to the customer's account from anyone else's bank deposit in excess of $10,000. If successful, such attacks would not only allow dishonest people to

steal, but could also undermine society's confidence in the integrity and reliability of the banking system.

Terrorists can also try to attack us through our computers. We cannot afford to have harmful computer programs destroy the computers driving the greater San Francisco metropolitan air traffic controller network, the New York Stock Exchange, the life support systems of a major hospital, or the Northern Virginia metropolitan area fire and paramedic emergency dispatch service.

There are many different kinds of "bad" computer programs, which in general are termed "Trojan horses" — programs that cause a computer to act in a manner not intended by its operator, named after the famous wooden horse of Troy that delivered an attacking army disguised as an attractive gift. One of the most notorious kinds is so-called "computer viruses" — "diseases" that a computer can "catch" from another computer. A computer virus is a computer program that instructs the computer to do harmful or spurious things instead of useful things — and can also replicate itself to spread from one computer to another. Since the computer does whatever its

instructions tell it to do, it will carry out the bad intent of a malicious human programmer who wrote the computer virus program — unless the computer is protected from the computer virus program. Special "anti-virus" protection software exists, but it unfortunately is only partially effective — for example, because new viruses can escape detection until they become widely known and recognized, and because sophisticated viruses can escape detection by masquerading as tasks the computer is supposed to be performing.

Computer security risks of all sorts — including the risks from computer viruses — have increased dramatically as computers have become increasingly connected to one another over the Internet and by other means. Increased computer connectivity provides increased capabilities, but also creates a host of computer security problems that haven't been fully solved. For example, electronic networks are an obvious path for spreading computer viruses. In October 1988, a university student used the Internet (a network of computer networks connected to millions of computers worldwide) to infect thousands of university and business computers with a self-replicating "worm"

virus that took over the infected computers and caused them to execute the computer virus instead of performing the tasks they were supposed to perform. This computer virus outbreak (which resulted in a criminal prosecution) caused widespread panic throughout the electronic community.

Computer viruses are by no means the only computer security risk made even more significant by increased computer connectivity. For example, a significant percentage of the online electronic community has recently become committed to a new "portable" computer language called Java™ developed by Sun Microsystems of Mountain View, California. Java was designed to allow computers to interactively and dynamically download computer program code fragments (called "applets") over an electronic network such as the internet, and execute the downloaded code fragments locally. Java's "download and execute" capability is valuable because it allows certain tasks to be performed locally on local equipment using local resources. For example, a user's computer could run a particularly computationally or data-intensive routine — relieving the provider's

computer from having to run the task and/or eliminating the need to transmit large amounts of data over the communications path.

While Java's "download and execute" capability has great potential, it raises significant computer security concerns. For example, Java applets could be written to damage hardware, software or information on the recipient computer, make the computer unstable by depleting its resources, and/or access confidential information on the computer and send it to someone else without first getting the computer owner's permission. People have expended lots of time and effort trying to solve Java's security problems. To alleviate some of these concerns, Sun Microsystems has developed a Java interpreter providing certain built-in security features such as:

- a Java verifier that will not let an applet execute until the verifier verifies the applet doesn't violate certain rules,

- a Java class loader that treats applets originating remotely differently from those originating locally,

- a Java security manager that controls access to resources such as files and network access, and

- promised to come soon — the use of digital signatures for authenticating applets.

Numerous security flaws have been found despite these techniques. Moreover, a philosophy underlying this overall security design is that a user will have no incentive to compromise the security of her own locally installed Java interpreter — and that any such compromise is inconsequential from a system security standpoint because only the user's own computer (and its contents) are at risk. This philosophy — which is typical of many security system designs — is seriously flawed in many useful electronic commerce contexts for reasons described below in connection with the above-referenced Ginter et al. patent specification.

The Ginter et al. specification describes a "virtual distribution environment" comprehensively providing overall systems and wide arrays of methods, techniques, structures and arrangements that

enable secure, efficient electronic commerce and rights management, including on the Internet or other "Information Super Highway."

The Ginter et al. patent disclosure describes, among other things, techniques for providing a secure, tamper resistant execution spaces within a "protected processing environment" for computer programs and data. The protected processing environment described in Ginter et al. may be hardware-based, software-based, or a hybrid. It can execute computer code the Ginter et al. disclosure refers to as "load modules." See, for example, Ginter et al. Figure 23 and corresponding text. These load modules — which can be transmitted from remote locations within secure cryptographic wrappers or "containers" — are used to perform the basic operations of the "virtual distribution environment." Load modules may contain algorithms, data, cryptographic keys, shared secrets, and/or other information that permits a load module to interact with other system components (e.g., other load modules and/or computer programs operating in the same or different protected processing environment). For a load module to operate and interact as intended, it must execute

without unauthorized modification and its contents may need to be protected from disclosure.

Unlike many other computer security scenarios, there may be a significant incentive for an owner of a Ginter et al. type protected processing environment to attack his or her own protected processing environment. For example:

- the owner may wish to "turn off" payment mechanisms necessary to ensure that people delivering content and other value receive adequate compensation; or

- the owner may wish to defeat other electronic controls preventing him or her from performing certain tasks (for example, copying content without authorization); or

- the owner may wish to access someone else's confidential information embodied within electronic controls present in the owner's protected processing environment; or

- the owner may wish to change the identity of a payment

recipient indicated within controls such that they receive payments themselves, or to interfere with commerce; or

- the owner may wish to defeat the mechanism(s) that disable some or all functions when budget has been exhausted, or audit trails have not been delivered.

Security experts can often be heard to say that to competently do their job, they must "think like an attacker." For example, a successful home security system installer must try to put herself in the place of a burglar trying to break in. Only by anticipating how a burglar might try to break into a house can the installer successfully defend the house against burglary. Similarly, computer security experts must try to anticipate the sorts of attacks that might be brought against a presumably secure computer system.

From this "think like an attacker" viewpoint, introducing a bogus load module is one of the strongest possible forms of attack (by a protected processing environment user or anyone else) on the virtual distribution environment disclosed in the Ginter et al. patent

specification. Because load modules have access to internal protected data structures within protected processing environments and also (at least to an extent) control the results brought about by those protected processing environments, bogus load modules can (putting aside for the moment additional possible local protections such as addressing and/or ring protection and also putting aside system level fraud and other security related checks) perform almost any action possible in the virtual distribution environment without being subject to intended electronic controls. Especially likely attacks may range from straightforward changes to protected data (for example, adding budget, billing for nothing instead of the desired amount, etc.) to wholesale compromise (for example, using a load module to expose a protected processing environment's cryptographic keys). For at least these reasons, the methods for validating the origin and soundness of a load module are critically important.

The Ginter et al. patent specification discloses important techniques for securing protected processing environments against inauthentic load modules introduced by the computer owner, user, or

any other party, including for example:

- Encrypting and authenticating load modules whenever they are shared between protected processing environments via a communications path outside of a tamper-resistant barrier and/or passed between different virtual distribution environment participants;

- Using digital signatures to determine if load module executable content is intact and was created by a trusted source (i.e., one with a correct certificate for creating load modules);

- Strictly controlling initiation of load module execution by use of encryption keys, digital signatures and/or tags;

- Carefully controlling the process of creating, replacing, updating or deleting load modules; and

- Maintaining shared secrets (e.g., cryptographic keys) within a tamper resistant enclosure that the owner of the electronic

appliance cannot easily tamper with.

Although the Ginter et al. patent specification comprehensively solves a host of load module (and other) security related problems, any computer system — no matter how secure — can be "cracked" if enough time, money and effort is devoted to the project. Therefore, even a very secure system such as that disclosed in Ginter et al. can be improved to provide even greater security and protection against attack.

The present invention provides improved techniques for protecting secure computation and/or execution spaces (as one important but non-limiting example, the protected processing environments as disclosed in Ginter et al) from unauthorized (and potentially harmful) load modules or other "executables" or associated data. In one particular preferred embodiment, these techniques build upon, enhance and/or extend in certain respects, the load module security techniques, arrangements and systems provided in the Ginter et al. specification.

In accordance with one aspect provided by the present invention, one or more trusted verifying authorities validate load modules or other executables by analyzing and/or testing them. A verifying authority digitally "signs" and "certifies" those load modules or other executables it has verified (using a public key based digital signature and/or certificate based thereon, for example).

Protected execution spaces such as protected processing environments can be programmed or otherwise conditioned to accept only those load modules or other executables bearing a digital signature/certificate of an accredited (or particular) verifying authority. Tamper resistant barriers may be used to protect this programming or other conditioning. The assurance levels described below are a measure or assessment of the effectiveness with which this programming or other conditioning is protected.

A web of trust may stand behind a verifying authority. For example, a verifying authority may be an independent organization that can be trusted by all electronic value chain participants not to collaborate with any particular participant to the disadvantage of

other participants. A given load module or other executable may be independently certified by any number of authorized verifying authority participants. If a load module or other executable is signed, for example, by five different verifying authority participants, a user will have (potentially) a higher likelihood of finding one that they trust. General commercial users may insist on several different certifiers, and government users, large corporations, and international trading partners may each have their own unique "web of trust" requirements. This "web of trust" prevents value chain participants from conspiring to defraud other value chain participants.

In accordance with another aspect provided by this invention, each load module or other executable has specifications associated with it describing the executable, its operations, content, and functions. Such specifications could be represented by any combination of specifications, formal mathematical descriptions that can be verified in an automated or other well-defined manner, or any other forms of description that can be processed, verified, and/or tested in an automated or other well-defined manner. The load

module or other executable is preferably constructed using a programming language (e.g., languages such as Java and Python) and/or design/implementation methodology (e.g., Gypsy, FDM) that can facilitate automated analysis, validation, verification, inspection, and/or testing.

A verifying authority analyzes, validates, verifies, inspects, and/or tests the load module or other executable, and compares its results with the specifications associated with the load module or other executable. A verifying authority may digitally sign or certify only those load modules or other executables having proper specifications — and may include the specifications as part of the material being signed or certified.

A verifying authority may instead, or in addition, selectively be given the responsibility for analyzing the load module and generating a specification for it. Such a specification could be reviewed by the load module's originator and/or any potential users of the load module.

A verifying authority may selectively be given the authority to generate an additional specification for the load module, for example by translating a formal mathematical specification to other kinds of specifications. This authority could be granted, for example, by a load module originator wishing to have a more accessible, but verified (certified), description of the load module for purposes of informing other potential users of the load module.

Additionally, a verifying authority may selectively be empowered to modify the specifications to make it accurate — but may refuse to sign or certify load modules or other executables that are harmful or dangerous irrespective of the accuracy of their associated specifications. The specifications may in some instances be viewable by ultimate users or other value chain participants — providing a high degree of assurance that load modules or other executables are not subverting the system and/or the legitimate interest of any participant in an electronic value chain the system supports.

In accordance with another aspect provided by the present invention, an execution environment protects itself by deciding — based on digital signatures, for example — which load modules or other executables it is willing to execute. A digital signature allows the execution environment to test both the authenticity and the integrity of the load module or other executables, as well permitting a user of such executables to determine their correctness with respect to their associated specifications or other description of their behavior, if such descriptions are included in the verification process.

A hierarchy of assurance levels may be provided for different protected processing environment security levels. Load modules or other executables can be provided with digital signatures associated with particular assurance levels. Appliances assigned to particular assurance levels can protect themselves from executing load modules or other executables associated with different assurance levels. Different digital signatures and/or certificates may be used to distinguish between load modules or other executables intended for different assurance levels. This strict assurance level hierarchy

provides a framework to help ensure that a more trusted environment can protect itself from load modules or other executables exposed to environments with different work factors (e.g., less trusted or tamper resistant environments). This can be used to provide a high degree of security compartmentalization that helps protect the remainder of the system should parts of the system become compromised.

For example, protected processing environments or other secure execution spaces that are more impervious to tampering (such as those providing a higher degree of physical security) may use an assurance level that isolates it from protected processing environments or other secure execution spaces that are relatively more susceptible to tampering (such as those constructed solely by software executing on a general purpose digital computer in a non-secure location).

A verifying authority may digitally sign load modules or other executables with a digital signature that indicates or implies assurance level. A verifying authority can use digital signature techniques to distinguish between assurance levels. As one example,

each different digital signature may be encrypted using a different verification key and/or fundamentally different encryption, one-way hash and/or other techniques. A protected processing environment or other secure execution space protects itself by executing only those load modules or other executables that have been digitally signed for its corresponding assurance level.

The present invention may use a verifying authority and the digital signatures it provides to compartmentalize the different electronic appliances depending on their level of security (e.g., work factor or relative tamper resistance). In particular, a verifying authority and the digital signatures it provides isolate appliances with significantly different work factors — preventing the security of high work factor appliances from collapsing into the security of low work factor appliances due to free exchange of load modules or other executables.

Encryption can be used in combination with the assurance level scheme discussed above to ensure that load modules or other executables can be executed only in specific environments or types of

environments. The secure way to ensure that a load module or other executable can't execute in a particular environment is to ensure that the environment doesn't have the key(s) necessary to decrypt it. Encryption can rely on multiple public keys and/or algorithms to transport basic key(s). Such encryption protects the load module or other executable from disclosure to environments (or assurance levels of environments) other than the one it is intended to execute in.

In accordance with another aspect provided by this invention, a verifying authority can digitally sign a load module or other executable with several different digital signatures and/or signature schemes. A protected processing environment or other secure execution space may require a load module or other executable to present multiple digital signatures before accepting it. An attacker would have to "break" each (all) of the several digital signatures and/or signature schemes to create an unauthorized load module or other executable that would be accepted by the protected processing environment or other secure execution space. Different protected processing environments (secure execution spaces) might examine

different subsets of the multiple digital signatures — so that compromising one protected processing environment (secure execution space) will not compromise all of them. As an optimization, a protected processing environment or other secure execution space might verify only one of the several digital signatures (for example, chosen at random each time an executable is used) — thereby speeding up the digital signature verification while still maintaining a high degree of security.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features and advantages provided in accordance with this invention may be better and more completely understood by referring to the following detailed description of example preferred embodiments in conjunction with the drawings, of which:

Figure 1 illustrates how defective or bogus load modules can wreak havoc in the electronic community;

Figure 2 shows an example verification authority that protects the electronic community from unauthorized load modules;

Figure 3 shows how a protected processing environment can distinguish between load modules that have been approved by a verifying authority and those that have not been approved;

Figure 4 shows an example process a verifying authority may perform to authenticate load modules;

Figure 5 shows how a verifying authority can create a certifying digital signature;

Figure 6 shows how a protected processing environment can securely authenticate a verifying authority's digital signature to guarantee the integrity of the corresponding load module;

Figure 7 shows how several different digital signatures can be applied to the same load module;

Figure 8 shows how a load module can be distributed with multiple digital signatures;

Figure 8A shows how key management can be used to compartmentalize protected processing environments;

Figures 9 shows how a load module can be segmented and each segment protected with a different digital signature;

Figures 10A-10C show how different assurance level electronic appliances can be provided with different cryptographic keys for authenticating verifying authority digital signatures;

Figures 11A-11C show how a verifying authority can use different digital signatures to designate the same or different load modules as being appropriate for execution by different assurance level electronic appliances;

Figures 12, 13 and 13A show how assurance level digital signatures can be used to isolate electronic appliances or appliance types based on work factor and/or tamper resistance to reduce overall

security risks; and

Figure 14 shows example overall steps that may be performed within an electronic system (such as, for example, a virtual distribution environment) to test, certify, distribute and use executables.

DETAILED DESCRIPTION OF EXAMPLE EMBODIMENTS

Figure 1 shows how defective, bogus and/or unauthorized computer information can wreak havoc within an electronic system 50. In this example, provider 52 is authorized to produce and distribute "load modules" 54 for use by different users or consumers 56. Figure 1 shows "load module" 54 as a complicated looking machine part for purposes of illustration only; the load module preferably comprises one or more computer instructions and/or data elements used to assist, allow, prohibit, direct, control or facilitate at least one task performed at least in part by an electronic appliance such as a computer. For example, load module 54 may comprise all or part of an executable computer program and/or associated data ("executable"), and may constitute a sequence of instructions or steps

that bring about a certain result within a computer or other computation element.

Figure 1 shows a number of electronic appliances 61 such as, for example, a set top box or home media player 58, a personal computer 60, and a multi-media player 62. Each of appliances 58, 60, 62 may include a secure execution space. One particular example of a secure execution space is a "protected processing environment" 108 of the type shown in Ginter et al. (see Figures 6-12) and described in associated text. Protected processing environments 108 provide a secure execution environment in which appliances 58, 60, 62 may securely execute load modules 54 to perform useful tasks. For example:

- Provider 52 might produce a load module 54a for use by the protected processing environment 108A within set top box or home media player 58. Load module 54a could, for example, enable the set top box/home media player 58 to play a movie, concert or other interesting program, charge users 56a a "pay per view" fee, and ensure that the fee is

paid to the appropriate rights holder (for example, the film studio, concert promoter or other organization that produced the program material).

- Provider 52 might produce another load module 54b for delivery to personal computer 60's protected processing environment 108B. The load module 54b might enable personal computer 60 to perform a financial transaction, such as, for example, home banking, a stock trade or an income tax payment or reporting.

- Provider 52 could produce a load module 54c for delivery to multi-media player 62's protected processing environment 108c. This load module 54c might allow user 56c to view a particular multi-media presentation while preventing the user from making a copy of the presentation—or it could control a portion of a transaction (e.g. a meter that records usage, and is incorporated into a larger transaction involving other load modules associated with interacting with a multi-media piece). (As described in

the Ginter et al. specification, load modules associated with the financial portion of a transaction, for example, may often be self contained and independent).

Figure 1 also shows an unauthorized and/or disreputable load module provider 64. Unauthorized provider 64 knows how to make load modules that look a lot like the load modules produced by authorized load module provider 52 — but are defective or even destructive. Unless precautions are taken, the unauthorized load module 54d made by unauthorized producer 64 will be able to run on protected processing environments 108 within appliances 58, 60 and 62, and may cause serious harm to users 56 and/or to the integrity of system 50. For example:

- unauthorized provider 64 could produce a load module 54d that is quite similar to authorized load module 54a intended to be used by set top box or home media player 58. The unauthorized load module 54d might allow protected processing environment 108A within set top box/home media player 58 to present the very same program material

— but divert some or all of the user's payment to unauthorized producer 64 — thereby defrauding the rights holders in the program material the users watch.

- Unauthorized provider 64 might produce an unauthorized version of load module 54b that could, if run by personal computer 60's protected processing environment 108b, disclose the user 64b's bank and credit card account numbers to unauthorized provider 64 and/or divert electronic or other funds to the unauthorized provider.

- Unauthorized provider 64 could produce an unauthorized version of load module 54c that could damage the protected processing environment 108c within multi media player 62 — erasing data it needs for its operation and making it unusable. Alternatively, an unauthorized version of load module 54c could defeat the copy protection provided by multi media player 62's protected processing environment, causing the makers of multi media programs to lose substantial revenues through unauthorized copying — or

defeat or alter the part of the transaction provided by the load module (e.g., billing, metering, maintaining an audit trail, etc.)

Figure 2 shows how a verifying authority 100 can prevent the problems shown in Figure 1. In this example, authorized provider 52 submits load modules 54 to verifying authority 100. Verifying authority 100 carefully analyzes the load modules 54 (see 102), testing them to make sure they do what they are supposed to do and do not compromise or harm system 50. If a load module 54 passes the tests verifying authority 100 subjects it to, a verifying authority may affix a digital "seal of approval" (see 104) to the load module.

Protected processing environments 108 can use this digital "seal of approval" 106 (which may comprise one or more "digital signatures") to distinguish between authorized and unauthorized load modules 54. Figure 3 illustrates how an electronic protected processing environment 108 can use and rely on a verifying authority's digital seal of approval 106. In this example, the protected processing environment 108 can distinguish between

authorized and unauthorized load modules 54 by examining the load module to see whether it bears the seal of verifying authority 100. Protected processing environment 108 will execute the load module 54a with its processor 110 only if the load module bears a verifying authority's seal 106. Protected processing environment 108 discards and does not use any load module 54 that does not bear this seal 106. In this way, protected processing environment 108 securely protects itself against unauthorized load modules 54 such as, for example, the defective load module 54d made by disreputable load module provider 64.

Figure 4 shows the analysis and digital signing steps 102, 104 performed by verifying authority 100 in this example. Provider 54 may provide, with each load module 54, associated specifications 110 identifying the load module and describing the functions the load module performs. In this example, these specifications 110 are illustrated as a manufacturing tag, but preferably comprises a data file associated with and/or attached to the load module 54.

Verifying authority 100 uses an analyzing tool(s) 112 to analyze and test load module 54 and determine whether it performs as specified by its associated specifications 110 — that is, whether the specifications are both accurate and complete. Figure 4 illustrates an analysis tool 112 as a magnifying glass; verifying authority 100 may not rely on visual inspection only, but instead preferably uses one or more computer-based software testing techniques and/or tools to verify that the load module performs as expected, matches specifications 110, is not a "virus," and includes no significant detectable "bugs" or other harmful functionality. See for example Pressman, Software Engineering: A Practitioner's Approach (3d Ed., McGraw-Hill 1992) at chapters 18 and 19 ("Software Testing Techniques") (pages 595-661) and the various books and papers referenced there. Although it has been said that "testing can show only the presence of bugs, not their absence," such testing (in addition to ensuring that the load module 54 satisfies its specifications 110) can provide added degrees of assurance that the load module isn't harmful and will work as it is supposed to.

Verifying authority 100 is preferably a trusted, independent third party such as an impartial, well respected independent testing laboratory. Therefore, all participants in an electronic transaction involving load module 54 can trust a verifying authority 100 as performing its testing and analysis functions competently and completely objectively and impartially. As described above, there may be several different verifying authorities 100 that together provide a "web of trust". Several different verifying authorities may each verify and digitally sign the same load module — increasing the likelihood that a particular value chain participant will trust one of them and decreasing the likelihood of collusion or fraud. Electronic value chain participants may rely upon different verifying authorities 100 to certify different types of load modules. For example, one verifying authority 100 trusted by and known to financial participants might verify load modules relating to financial aspects of a transaction (e.g., billing), whereas another verifying authority 100' trusted by and known to participants involved in using the "information exhaust" provided by an electronic transaction might be used to verify load modules relating to usage metering aspects of the

same transaction.

Once verifying authority 100 is satisfied with load module 54, it affixes its digital "seal of approval" 106 to the load module. Figure 4 illustrates the digital sealing process as being performed by a stamp 114 — but in the preferred embodiment the digital sealing process is actually performed by creating a "digital signature" using a well known process. See Schneier, Applied Cryptography (2d Ed. John Wiley & Sons 1996) at Chapter 20 (pages 483-502). This digital signature, certificate or seal creation process is illustrated in Figure 5.

In the Figure 5 process, load module 54 (along with specifications 110 if desired) is processed to yield a "message digest" 116 using a conventional one-way hash function selected to provide an appropriate resistance to algorithmic attack. See, for example, the transformation processes discussed in the Schneier text at Chapter 18, pages 429-455. A one-way hash function 115 provides a "fingerprint" (message digest 116) that is unique to load module 54. The one-way hash function transforms the contents of load module 54 into message digest 116 based on a mathematical function. This

one-way hash mathematical function has the characteristic that it is easy to calculate message digest 116 from load module 54, but it is hard (computationally infeasible) to calculate load module 54 starting from message digest 116 and it is also hard (computationally infeasible) to find another load module 54' that will transform to the same message digest 116. There are many potential candidate functions (e.g., MD5, SHA), families of functions (e.g., MD5, or SHA with different internal constants), and keyed functions (e.g., message authentication codes based on block ciphers such as DES) that may be employed as one-way hash functions in this scheme. Different functions may have different cryptographic strengths and weaknesses so that techniques which may be developed to defeat one of them are not necessarily applicable to others.

Message digest 116 may then be encrypted using asymmetric key cryptography. Figure 5 illustrates this encryption operation using the metaphor of a strong box 118. The message digest 116 is placed into strong box 118, and the strongbox is locked with a lock 120 having two key slots opened by different ("asymmetrical") keys.

A first key 122 (sometimes called the "private" key) is used to lock the lock. A second (different) key 124 (sometimes called the "public" key) must be used to open the lock once the lock has been locked with the first key. The encryption algorithm and key length is selected so that it is computationally infeasible to calculate first key 122 given access to second key 124, the public key encryption algorithm, the clear text message digest 116, and the encrypted digital signature 106. There are many potential candidate algorithms for this type of asymmetric key cryptography (e.g., RSA, DSA, El Gamal, Elliptic Curve Encryption). Different algorithms may have different cryptographic strengths and weaknesses so that techniques which may be developed to defeat one of them are not necessarily applicable to others.

In this case the first key is owned by verifying authority 100 and is kept highly secure (for example, using standard physical and procedural measures typically employed to keep an important private key secret while preventing it from being lost). Once message digest 116 is locked into strong box 118 using the first key 122 the strong

box can be opened only by using the corresponding second key 124. Note that other items (e.g., further identification information, a time/date stamp, etc.) can also be placed within strong box 106.

Figure 6 shows how a protected processing environment 108 "authenticates" the digital signature 106 created by the Figure 5 process. Second key 124 and the one-way hash algorithm are first securely provided to the protected processing environment. For example, a secure key exchange protocol can be used as described in connection with Figure 64 of the Ginter et al. patent specification. Public key cryptography allows second key 124 to be made public without compromising first key 122. However, in this example, protected processing environment 108 preferably keeps the second key 124 (and, if desired, also the one-way hash algorithm and/or its associated key) secret to further increase security.

Maintaining "public" verification key 124 as a secret within tamper resistant protected processing environment 108 greatly complicates the job of generating bogus digital signatures 106. If the attacker does not possess second key 124, the difficulty of an

algorithmic attack or cryptanalytic attack on the verification digital signature algorithm is significantly increased, and the attacker might be reduced to exhaustive search (brute force) type attacks which would be even less practical because the search trials would require attempting to present a bogus load module 54 to protected processing environment 108 — which, after a few such attempts is likely to refuse all further attempts. Keeping second key 124 secret also requires a multi-disciplinary attack: an attacker must both (A) extract the secret from protected processing environment 108, and (B) attack the algorithm. It may be substantially less likely that a single attacker may have expertise in each of these two specialized disciplines.

In addition, maintaining the "public" key within a tamper-resistant environment forecloses the significant threat that the owner of protected processing environment 108 may himself attack the environment. For example, if the owner could replace the appropriate "public" key 124 with his own substitute public key, the owner could force the protected processing environment 108 to

execute load modules 54 of his own design — thereby compromising the interests of others in enforcing their own controls within the owner's protected processing environment. For example, the owner could turn off the control that required him to pay for watching or prohibited him from copying content. Since protected processing environment 108 can support a "virtual business presence" by parties other than the owner, it is important for the protected processing environment to be protected against attacks from the owner.

The load module 54 and its associated digital signature 106 is then delivered to the protected processing environment 108. (These items can be provided together at the same time, independently, or at different times.) Protected processing environment 115 applies the same one way hash transformation on load module 54 that a verifying authority 100 applied. Since protected processing environment 108 starts with the same load module 54 and uses the same one-way hash function 115, it should generate the same message digest 116'.

Protected processing environment 108 then decrypts digital signature 106 using the second key 124 — i.e., it opens strongbox

118 to retrieve the message digest 116 a verifying authority 100 placed in there. Protected processing environment 108 compares the version of message digest 116 it obtains from the digital signature 106 with the version of message digest 116' it calculates itself from load module 54 using the one way hash transformation 115. The message digests 116, 116' should be identical. If they do not match, digital signature 106 is not authentic or load module 54 has been changed — and protected processing environment 108 rejects load module 54.

Figure 7 shows that multiple digital signatures 106(1), 106(2), ... 106(N) can be created for the same load module 54. For example:

- one digital signature 106(1) can be created by encrypting message digest 116 with a "private" key 122(1),

- another (different) digital signature 106(2) can be created by encrypting the message digest 116 with a different "private" key 122(2), possibly employing a different signature algorithm, and

- a still different digital signature 106(N) can be generated by encrypting the message digest using a still different "private" key 122(N), possibly employing a different signature algorithm.

The public key 124(1) corresponding to private key 122(1) acts only to decrypt (authenticate) digital signature 106(1). Similarly, digital signature 106' can only be decrypted (authenticated) using public key 124(2) corresponding to the private 122(2). Public key 124(1) will not "unlock" digital signature 106(2) and public key 124(2) will not "unlock" digital signature 106(1).

Different digital signatures 106(1), 106(N) can also be made by using different one way hash functions 115 and/or different encryption algorithms. As shown in Figure 8, a load module 54 may have multiple different types of digital signatures 106 associated with it. Requiring a load module 54 to present, to a protected processing environment 108, multiple digital signatures 106 generated using fundamentally different techniques decreases the risk that an attacker can successfully manufacture a bogus load module 54.

For example, as shown in Figure 8, the same load module 54 might be digitally signed using three different private keys 122, cryptographic algorithms, and/or hash algorithms. If a given load module 54 has multiple distinct digital signatures 106 each computed using a fundamentally different technique, the risk of compromise is substantially lowered. A single algorithmic advance is unlikely to result in simultaneous success against both (or multiple) cryptographic algorithms. The two digital signature algorithms in widespread use today (RSA and DSA) are based on distinct mathematical problems (factoring in the case of RSA, discrete logs for DSA). The most currently popular one-way hash functions (MD4/MD5 and SHA) have similar internal structures, possibly increasing the likelihood that a successful attack against one would lead to a success against another. However, hash functions can be derived from any number of different block ciphers (e.g., SEAL, IDEA, triple-DES) with different internal structures; one of these might be a good candidate to complement MD5 or SHA.

Multiple signatures as shown in Figure 8 impose a cost of additional storage for the signatures 106 in each protected load module 54, additional code in the protected processing environment 108 to implement additional algorithms, and additional time to verify the digital signatures (as well as to generate them at verification time). As an optimization to the use of multiple keys or algorithms, an appliance 61 might verify only a subset of several signatures associated with a load module 54 (chosen at random) each time the load module is used. This would speed up signature verification while maintaining a high probability of detection. For example, suppose there are one hundred "private" verification keys, and each load module 54 carries one hundred digital signatures. Suppose each protected processing environment 108, on the other hand, knows only a few (e.g., ten) of these corresponding "public" verification keys randomly selected from the set. A successful attack on that particular protected processing environment 108 would permit it to be compromised and would also compromise any other protected processing environment possessing and using precisely that same set of ten keys. However, it would not compromise most other protected

processing environments — since they would employ a different subset of the keys used by verifying authority 100.

Figure 8A shows a simplified example of different processing environments 108(1), ... , 108(N) possessing different subsets of "public" keys used for digital signature authentication — thereby compartmentalizing the protected processing environments based on key management and availability. The Figure 8A illustration shows each protected processing environment 108 having only one "public" key 124 that corresponds to one of the digital signatures 106 used to "sign" load module 54. As explained above, any number of digital signatures 106 may be used to sign the load module 54 — and different protected processing environment 108 may possess any subset of corresponding "public" keys.

Figure 9 shows that a load module 54 may comprise multiple segments 55(1), 55(2), 55(3) signed using different digital signatures 106. For example:

- a first load module segment 55(1) might be signed using a

digital signature 106(1);

- a second load module segment 55(2) might be digitally signed using a second digital signature 106(2); and

- a third load module segment 55(3) might be signed using a third digital signature 106(3).

These three signatures 55(1), 55(2), 55(3) could all be affixed by the same verifying authority 100, or they could be affixed by three different verifying authorities (providing a "web of trust"). (In another model, a load module is verified in its entirety by multiple parties -- if a user trusts any of them, she can trust the load module.) A protected processing environment 108 would need to have all three corresponding "public" keys 124(1), 124(2), 124(3) to authenticate the entire load module 54 — or the different load module segments could be used by different protected processing environments possessing the corresponding different keys 124(1), 124(2), 124(3). Different signatures 55(1), 55(2), 55(3) could be calculated using different signature and/or one-way hash algorithms to increase the

difficulty of defeating them by cryptanalytic attack.

Assurance Levels

Verifying authority 100 can use different digital signing techniques to provide different "assurance levels" for different kinds of electronic appliances 61 having different "work factors" or levels of tamper resistance. Figures 10A-10C show an example assurance level hierarchy providing three different assurance levels for different electronic appliance types:

- Assurance level I might be used for an electronic appliance(s) 61 whose protected processing environment 108 is based on software techniques that may be somewhat resistant to tampering. An example of an assurance level I electronic appliance 61A might be a general purpose personal computer that executes software to create protected processing environment 108.

- An assurance level II electronic appliance 61B may provide a protected processing environment 108 based on a hybrid

of software security techniques and hardware-based security techniques. An example of an assurance level II electronic appliance 61B might be a general purpose personal computer equipped with a hardware integrated circuit secure processing unit ("SPU") that performs some secure processing outside of the SPU (see Ginter et al. patent disclosure Figure 10 and associated text). Such a hybrid arrangement might be relatively more resistant to tampering than a software-only implementation.

- The assurance level III appliance 61C shown is a general purpose personal computer equipped with a hardware-based secure processing unit 132 providing and completely containing protected processing environment 108 (see Ginter et al. Figures 6 and 9 for example). A silicon-based special purpose integrated circuit security chip is relatively more tamper-resistant than implementations relying on software techniques for some or all of their tamper-resistance.

In this example, verifying authority 100 digitally signs load modules 54 using different digital signature techniques (for example, different "private" keys 122) based on assurance level. The digital signatures 106 applied by verifying authority 100 thus securely encode the same (or different) load module 54 for use by appropriate corresponding assurance level electronic appliances 61.

Assurance level in this example may be assigned to a particular protected processing environment 108 at initialization (e.g., at the factory in the case of hardware-based secure processing units). Assigning assurance level at initialization time facilitates the use of key management (e.g., secure key exchange protocols) to enforce isolation based on assurance level. For example, since establishment of assurance level is done at initialization time, rather than in the field in this example, the key exchange mechanism can be used to provide new keys (assuming an assurance level has been established correctly).

Within a protected processing environment 108, as shown in Figures 10A-10C, different assurance levels may be assigned to each

separate instance of a channel (see Ginter et al., Figure 15) contained therein. In this way, each secure processing environment and host event processing environment (see Ginter et al., Figure 10 and associated description) contained within an instance of a PPE 108 may contain multiple instances of a channel, each with independent and different assurance levels. The nature of this feature of the invention permits the separation of different channels within a PPE 108 from each other, each channel possibly having identical, shared, or independent sets of load modules for each specific channel limited solely to the resources and services authorized for use by that specific channel. In this way, the security of the entire PPE is enhanced and the effect of security breaches within each channel is compartmentalized solely to that channel.

As shown in Figure 11A-11C, different digital signatures and/or signature algorithms corresponding to different "assurance levels" may be used to allow a particular execution environment to protect itself from particular load modules 54 that are accessible to other classes or "assurance levels" of electronic appliances. As

shown in Figures 11A-11C:

- A protected processing environment(s) of assurance level I protects itself (themselves) by executing only load modules 54 sealed with an assurance level I digital signature 106(I). Protected processing environment(s) 108 having an associated assurance level I is (are) securely issued a public key 124(I) that can "unlock" the level I digital signature.

- Similarly, a protected processing environment(s) of assurance level II protects itself (themselves) by executing only the same (or different) load module 54 sealed with a "Level II" digital signature 106(II). Such a protected processing environment 108 having an associated corresponding assurance level II possess a public key 124(II) used to "unlock" the level II digital signature.

- A protected processing environment(s) 108 of assurance level III protects itself (themselves) by executing only load modules 54 having a digital signature 106(III) for assurance level III. Such an assurance level III protected processing environment 108 possesses a corresponding assurance level 3 public key 124(III). Key management encryption (not signature) keys can allow this protection to work securely.

In this example, electronic appliances 61 of different assurance levels can communicate with one another and pass load modules 54 between one another — an important feature providing a scaleable virtual distribution environment involving all sorts of different appliances (e.g., personal computers, laptop computers, handheld computers, television sets, media players, set top boxes, internet browser appliances, smart cards, mainframe computers, etc.) The present invention uses verifying authority 100 and the digital signatures it provides to compartmentalize the different electronic

appliances depending on their level of security (e.g., work factor or relative tamper resistance). In particular, verifying authority 100 and the digital signatures it provides isolate appliances with significantly different work factors — preventing the security of high work factor appliances from collapsing into the security of low work factor appliances due to free exchange of load modules 54.

In one example, verifying authority 100 may digitally sign identical copies of load module 54 for use by different classes or "assurance levels" of electronic appliances 61. If the sharing of a load module 54 between different electronic appliances is regarded as an open communications channel between the protected processing environments 108 of the two appliances, it becomes apparent that there is a high degree of risk in permitting such sharing to occur. In particular, the extra security assurances and precautions of the more trusted environment are collapsed into the those of the less trusted environment because an attacker who compromises a load module within a less trusted environment is then be able to launch the same load module to attack the more trusted environment. Hence,

although compartmentalization based on encryption and key management can be used to restrict certain kinds of load modules 54 to execute only on certain types of electronic appliances 61, a significant application in this context is to compartmentalize the different types of electronic appliances and thereby allow an electronic appliance to protect itself against load modules 54 of different assurance levels.

Figure 12 emphasizes this isolation using the illustrative metaphor of desert islands. It shows how the assurance levels can be used to isolate and compartmentalize any number of different types of electronic appliances 61. In this example:

- Personal computer 60(1) providing a software-only protected processing environment 108 may be at assurance level I;

- Media player 400(1) providing a software-only based protected processing environment may be at assurance level II;

- Server 402(1) providing a software-only based protected processing environment may be at assurance level III;

- Support service 404(1) providing a software-only based protected processing environment may be at assurance level IV;

- Personal computer 60(2) providing a hybrid software and hardware protected processing environment 108 may be at assurance level V;

- Media player 400(2) providing a hybrid software and hardware protected processing environment may be at assurance level VI;

- Server 402(2) providing a software and hardware hybrid protected processing environment may be at assurance level VII;

- Support service 404(2) providing a software and hardware hybrid protected processing environment may be at

assurance level VIII; and

- Personal computer 60(3) providing a hardware-only protected processing environment 108 may be at assurance level IX;

- Media player 400(3) providing a hardware-only protected processing environment may be at assurance level X;

- Server 402(3) providing a hardware-only based protected processing environment may be at assurance level XI;

- Support service 404(3) providing a hardware-only based protected processing environment may be at assurance level XII.

In accordance with this feature of the invention, verifying authority 100 supports all of these various categories of digital signatures, and system 50 uses key management to distribute the appropriate verification keys to different assurance level devices. For example, verifying authority 100 may digitally sign a particular load

module 54 such that only hardware-only based server(s) 402(3) at assurance level XI may authenticate it. This compartmentalization prevents any load module executable on hardware-only servers 402(3) from executing on any other assurance level appliance (for example, software-only protected processing environment based support service 404(1)).

To simplify key management and distribution, execution environments having significantly similar work factors can be classified in the same assurance level. Figure 13 shows one example hierarchical assurance level arrangement. In this example, less secure "software only" protected processing environment 108 devices are categorized as assurance level I, somewhat more secure "software and hardware hybrid" protected processing environment appliances are categorized as assurance level II, and more trusted "hardware only" protected processing environment devices are categorized as assurance level III.

To show this type of isolation, Figure 13A shows three example corresponding "desert islands." Desert island I is

"inhabited" by personal computers 61A providing a software-only protected processing environment. The software-only protected processing environment based personal computers 60(1) "inhabit" desert island I are all of the same assurance level — and thus will each authenticate (and may thus each use) an assurance level I load module 54a. Desert island II is "inhabited" by assurance level II hybrid software and hardware protected processing environment personal computers 61B. These assurance level II personal computers will each authenticate (and may thus each execute) an assurance level II load module 54b. Similarly, a desert island III is "inhabited" by assurance level III personal computers 61C providing hardware-only protected processing environments. These assurance level III devices 61C may each authenticate and execute an assurance level III load module 54c.

The "desert islands" are created by the use of different digital signatures on each of load modules 54a, 54b, 54c. In this example, all of the appliances 61 may freely communicate with one another (as indicated by the barges — which represent electronic or other

communications between the various devices. However, because particular assurance level load modules 54 will be authenticated only by appliances 60 having corresponding assurance levels, the load modules cannot leave their associated "desert island" — providing isolation between the different assurance level execution environments. More specifically, a particular assurance level appliance 61 thus protects itself from using a load module 54 of a different assurance level. Digital signatures (and/or signature algorithms) 106 in this sense create the isolated "desert islands" shown — since they allow execution environments to protect themselves from "off island" load modules 54 of different assurance levels.

A load module or other executable may be certified for multiple assurance levels. Different digital signatures may be used to certify the same load module or other executable for different respective assurance levels. The load module or other executable could also be encrypted differently (e.g. using different keys to encrypt the load module) based on assurance level. If a load module

is encrypted differently for different assurance levels, and the keys and/or algorithms that are used to decrypt such load modules are only distributed to environments of the same assurance level, an additional measure of security is provided. The risk associated with disclosing the load module or other executable contents (e.g., by decrypting encrypted code before execution) in a lower assurance environment does not compromise the security of higher assurance level systems directly, but it may help the attacker learn how the load module or other executable works and how to encrypt them — which can be important in making bogus load modules or other executables (although not in certifying them — since certification requires keys that would only become available to an attacker who has compromised the keys of a corresponding appropriate assurance level environment). Commercially, it may be important for administrative ease and consistency to take this risk. In other cases, it will not be (e.g. provider sensitivities, government uses, custom functions, etc.)

Figure 14 shows an example sequence of steps that may be performed in an overall process provided by these inventions. To

begin the overall process, a load module provider 52 may manufacture a load module and associated specifications (Figure 14, block 502). Provider 52 may then submit the load module and associated specifications to verifying authority 100 for verification (Figure 14, block 504). Verifying authority 100 may analyze, test, and/or otherwise validate the load module against the specifications (Figure 14, block 506), and determine whether the load module satisfies the specifications.

If the load module is found to satisfy its specifications, a verifying authority 100 determines whether it is authorized to generate one or more new specifications for the load module (Figure 14, block 509). If it is authorized and this function has been requested ("Y" exit to decision block 509), a verifying authority generates specifications and associates them with the load module (Figure 14, block 514).

If the load module fails the test ("N" exit to decision block 508), verifying authority 100 determines whether it is authorized and able to create new specifications corresponding to the actual load

module performance, and whether it is desirable to create the conforming specifications (Figure 14, decision block 510). If verifying authority 100 decides not to make new specifications ("N" exit to decision block 510), verifying authority returns the load module to provider 52 (block 512) and the process ends. On the other hand, if verifying authority 100 determines that it is desirable to make new specifications and it is able and authorized to do so, a verifying authority 100 may make new specifications that conform to the load module ("Y" exit to decision block 510; block 514).

A verifying authority 100 may then digitally sign the load module 54 to indicate approval (Figure 14, block 516). This step 516 may involve applying multiple digital signatures and/or a selection of the appropriate digital signatures to use in order to restrict the load module to particular "assurance levels" of electronic appliances as discussed above. Verifying authority may then determine the distribution of the load module (Figure 14, block 518). This "determine distribution" step may involve, for example, determining who the load module should be distributed to (e.g., provider 52,

support services 404, a load module repository operated by a verifying authority, etc.) and/or what should be distributed (e.g., the load module plus corresponding digital signatures, digital signatures only, digital signatures and associated description, etc.). Verifying authority 100 may then distribute the appropriate information to a value chain using the appropriate distribution techniques (Figure 14, block 520).